COMPUTING SCIENCE

A Tool for the Automatic Verification of BPMN Choreographies

Ellis Solaiman, Wenzhong Sun, and Carlos Molina-Jimenez

TECHNICAL REPORT SERIES

No. CS-TR-1464 April 2015

TECHNICAL REPORT SERIES

No. CS-TR-1464 April, 2015

A Tool for the Automatic Verification of BPMN Choreographies

E. Solaiman, W. Sun, C. Molina-Jimenez

Abstract

The Business Process Model Notation (BPMN) provides a standard graphical language that can be used by business analysts for modeling business process choreographies. A challenging task is to formally verify that constructed choreography models are logically correct with respect to safety, liveness, and various application-specific correctness requirements. To aid with this important task, we present a model checker based framework to automate the verification process. The main component of our framework is the BPMNverifier, a tool that can automatically convert BPMN choreography models into PROMELA, the input language of the SPIN model checker. We describe the implementation and functionality of the BPMNverifier, and how the tool eases the task of expressing Linear Temporal Logic (LTL) correctness requirements, through its LTL Manager component.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

SOLAIMAN, E; SUN, W; MOLINA-JIMENEZ, C
A Tool for the Automatic Verification of BPMN Choreographies
[By] E. Solaiman, W. Sun, C. Molina-Jimenez
Newcastle upon Tyne: Newcastle University: Computing Science, 2015.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1464)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1464

Abstract

The Business Process Model Notation (BPMN) provides a standard graphical language that can be used by business analysts for modeling business process choreographies. A challenging task is to formally verify that constructed choreography models are logically correct with respect to safety, liveness, and various application-specific correctness requirements. To aid with this important task, we present a model checker based framework to automate the verification process. The main component of our framework is the BPMNverifier, a tool that can automatically convert BPMN choreography models into PROMELA, the input language of the SPIN model checker. We describe the implementation and functionality of the BPMNverifier, and how the tool eases the task of expressing Linear Temporal Logic (LTL) correctness requirements, through its LTL Manager component.

About the authors

Dr Ellis Solaiman received his PhD in the School of Computing Science at the University of Newcastle upon Tyne in 2004 for work on Electronic Contract specification, monitoring, and verification. He is currently a Teaching Fellow in the School of Computing Science, Newcastle University. His research interests are on Trust Management, Middleware Technologies, Modelling of Business Processes and Service Level Agreements (SLAs), SLA Monitoring, and Verification.

Mr Wenzhong Sun (Jim) completed an MSc Advanced Computer Science at Newcastle University with distinction in 2012.

Dr Carlos Molina-Jimenez received his PhD in the School of Computing Science at the University of Newcastle upon Tyne in 2000 for work on anonymous interactions on the Internet. He is currently a Research Associate at the Computer Laboratory at Cambridge University. His research interests are on Information Coordination and Sharing in Virtual Enterprises, Trust Management and Electronic Contracting.

Suggested keywords

BUSINESS PROCESSES
BPMN
CHOREOGRAPHIES
MODEL CHECKING
VERIFICATION.

A Tool for the Automatic Verification of BPMN Choreographies

Ellis Solaiman
School of Computing Science
Newcastle University
United Kingdom
ellis.solaiman@ncl.ac.uk

Wenzhong Sun
School of Computing Science
Newcastle University
United Kingdom
eatsun1983@gmail.com

Carlos Molina-Jimenez
The Computer Laboratory
University of Cambridge
United Kingdom
carlos.molina@cl.cam.ac.uk

***Abstract*—The Business Process Model Notation (BPMN) provides a standard graphical language that can be used by business analysts for modeling business process choreographies. A challenging task is to formally verify that constructed choreography models are logically correct with respect to safety, liveness, and various application-specific correctness requirements. To aid with this important task, we present a model checker based framework to automate the verification process. The main component of our framework is the BPMNverifier, a tool that can automatically convert BPMN choreography models into PROMELA, the input language of the SPIN model checker. We describe the implementation and functionality of the BPMNverifier, and how the tool eases the task of expressing Linear Temporal Logic (LTL) correctness requirements, through its LTL Manager component.**

***Keywords*-business processes, BPMN, choreographies, model checking, verification.**

I. Introduction

We consider Business to Business (B2B) interactions conducted over the Internet between two or more business partners. Such relationships normally involve the execution of one or more shared business processes (also known as public, global, or cross–organizational business processes). Each business partner is responsible for performing its part of the cross-organisational business process. Thus in a scenario of N business partners, the cross-organisational business process can be regarded as composed of N individual business processes (one within each business partner) that interact with each other by means of exchanging messages over communication channels.

Before individual business processes can be implemented, one must be able to compose the overall shared process in the form of a choreography. Naturally, a choreography specification that intends to capture the complexities of B2B cross-organisational interactions must be verified for correctness before it can be enacted by the individual business partners. The aim of this paper is to present a tool based framework through which choreography verification can be done automatically.

To illustrate let us take a look at a simple example. Below is a hypothetical business contract between a *Buyer* and *Store*. Before a B2B relationship can commence, normally a business contract needs to be negotiated and agreed. The clauses of the contract should take into consideration all business operations (shown in bold in the contract text) that are relevant to the execution of the shared business process.

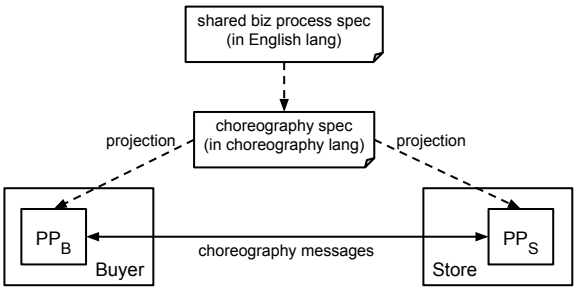

Figure 1. Choreography and public processes.

1) *The buyer can place a **buy request** with the store to buy an item.*
2) *The store is obliged to respond with either **buy confirmation** or **buy rejection** within 3 days of receiving the buy request.*
   a) *No response from the store within 3 days will be treated as a buy rejection.*
3) *The buyer can either **pay** or **cancel** the buy request within 7 days of receiving a confirmation.*
   a) *No response from the buyer within 7 days will be treated as a cancellation.*

Current industrial practice makes use of contracts implicitly in designing choreographies. The idea of explicitly using contracts in deriving choreographies and/or business processes of partners is a topic of ongoing research [1] [2].

Fig. 1 shows the two arbitrary organizations; *Buyer*, and *Store* that are interested in executing the shared business process. In the figure, the contract specification is used for producing the formal choreography specification. The enactment of the choreography results in two processes; PP_B and PP_S (*Public Process Buyer* and *Public Process Store*, respectively). The two processes interact with each other by means of exchanging messages over a communication channel represented by *choreography messages* in the figure.

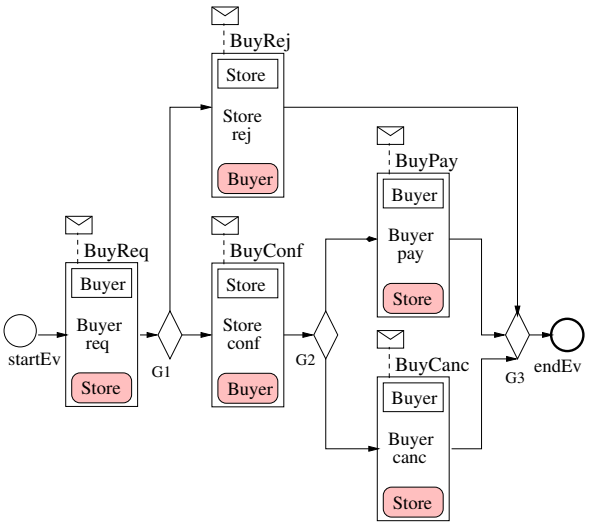

Figure 2. BPMN choreography between a buyer and store.

The choreography specification describes, formally, and from a global perspective, all permissible message exchange sequences between the business partners. An example choreography for our contract example can be seen in Fig. 2. We will describe this in detail in Section II. Because a choreography is a formal description, it can be used for producing the individual processes of the two parties by means of mechanical projection [3].

A key problem when producing a formal choreography specification from an informal specification such as a contract, is that the choreography specification will likely suffer from logical errors and inconsistencies. Naturally, detecting logical errors manually is a difficult and time consuming process that becomes more difficult as the choreograph becomes more complex. Therefore it becomes necessary to resort to rigorous formal verification techniques. A pragmatic approach to verifying the correctness of choreography specification is model checking.

Model checking techniques have reached maturity. Yet their application to choreography verification is not widely spread. We contend that two of the main difficulties that choreography designers face are; construction of the validation model, and specification of the choreography correctness properties. For example, in order to be able to use the model checking tool SPIN [4], a choreography designer needs to master PROMELA, the input language of SPIN, and LTL (Linear Temporal Logic), the language for expressing correctness properties [5]. Although PROMELA is a well documented language, building validation models in PROMELA is still time consuming and a distraction from the core task of building choreographies. Also, it is widely acknowledged that LTL is a powerful language for expressing correctness properties. Yet it has proven to be hard to master for non–experts in temporal logic. For instance, the LTL syntax traditionally accepted by SPIN is low level and based on the basic temporal logic operators (!, [], <>, etc.), which results in LTL formulae that are not easy to read or write. In addition, the semantics of LTL formulae are very subtle; thus writing an LTL formulae that captures the intended correctness requirement within a PROMELA model is particularly challenging and error prone.

In order to meet these challenges, and to aid choreography designers in the process of verifying the logical correctness of choreography specifications, we have implemented a BPMN verification tool. The *BPMNverifier* is a graphical tool implemented in Java, which can automatically convert choreography specifications written in BPMN 2.0 (Business Process Management Notation) [6] into PROMELA models. BPMN is a well known standard graphical notation that is widely used for specifying business processes and choreographies. In addition, the *BPMNverifier* provides an *LTL Manager* component, and a repository that can be populated by LTL experts with LTL templates (LTL formulae with abstract variables) of typical correctness properties required for business choreographies, together with their English language descriptions. These LTL templates can be easily accessed and parameterized by choreography designers to produce specific LTL correctness properties. The LTL properties are then mechanically included in the automatically generated PROMELA models and presented to SPIN for verification. The aim of this paper is to present the architecture of the *BPMNverifier*, the *LTL Manager*, and their implementation.

The remainder of the paper is organised as follows: We continue our discussion in Section II with a summary of the concepts and technologies involved in our work. The architecture and functionality of the tool is discussed in Section III. Section IV demonstrates the use of the tool by means of an example. We place our tool within the context of current research in section V. We draw conclusions and motivate further research in this direction in Section VI.

II. Background

A. BPMN choreography diagrams

BPMN is standard language for modelling business processes and is maintained by the OMG (Object Management Group) [6]. The original specification aims at generality and includes a rich set of constructs that can be used for specifying process models at different levels of abstraction. The focus of our current research is on choreography specifications. Therefore in this paper, we will adopt the RosettaNet version of BPMN as specified in [7], which is aimed at the choreography level of abstraction; thus it is comparatively smaller and simpler than the original BPMN specification language. Like the original BPMN, RosettaNet BPMN specifies a set of constructs for capturing interactions between business partners at the choreography level.

To explain the BPMN constructs supported by the BPMNverifier, we will use the choreography diagram of Fig. 2, which is a possible specification of the business contract example discussed in Section I:

Events: are represented using circles, thus *startEv* and *endEv* represent, respectively, the start and end events of the process. **Activities**: are represented by a box that specifies the name of the activity, participants, and messages. The figure includes five activities called *Buyer req, Store rej, Store conf, Buyer pay* and *Buyer canc*. They represent, respectively; the placement of the Buyer's buy request, the Store's rejection of the request, the Store's confirmation of the request, the Buyer's submission of payment, and the Buyer's cancellation of the request. **Participants**: the names of the participants in each activity are specified inside bands of different colours. The sender in a white band and the receiver in a shaded band. **Messages**: include the information exchanged between two participants in an activity and are represented by envelops. The figure includes five messages, namely, *BuyReq, BuyRej, BuyConf, BuyPay* and *BuyCanc*, which stand for, respectively, *BuyRequest, BuyRejection, BuyConfirmation, BuyPayment* and *BuyCancellation*. For example, in the *Buyer req* activity, the *Buyer* sends the *BuyReq* message to the *Store*. **Sequence flows**: represented by arrowed lines, and indicate the order of execution of activities. **Gateways**: model split and join points in execution flows and are represented by diamonds. The figure includes two exclusive fork gateways (*G1* and *G2*) and a single exclusive merge one (*G3*).

B. The SPIN Model Checker and its Input Language PROMELA

SPIN is a model checker designed for reasoning about the logical correctness of distributed systems composed of several processes that execute asynchronously (exactly one process can make a transition at a time) following the interleaving model of concurrent execution [4]. We use it in our research because it is currently one of the most mature, well documented and widely available model checkers [8]. More importantly, it meets the technical requirements that we need for the validation of choreography diagrams. SPIN can verify safety and liveness properties of abstract models (validation models) written in its input language, called PROMELA. When no errors are detected by SPIN, the output is simply *errors: 0* (in addition to some statistics about the verification run). Conversely, when SPIN detects an error, it stops the verification run and produces a counter example (a **.trail* file on disk). Several counter examples can be produced by manipulation of configuration options.

The three basic building blocks of a validation model written in PROMELA are processes, buffered channels, and variables. A validation model normally contains two or more user–defined processes that communicate with each other by means of sending and receiving messages over channels with zero or more slots in their buffers. In addition, it includes an `init` process that is used for initializing the component processes. A concise summary of PROMELA can be found in [9]. We will briefly discuss here only the constructs involved in our BPMN to PROMELA translator:

User–defined processes: are the executable entities in a validation model and declared by the keyword *proctype*. For example *proctype Buyer(...)* {...} declares a process called *Buyer*. **Init process**: is used for initializing the user–defined processes and declared by the keyword *init*. For instance, *init* {*run Buyer(...); Store(...)*} initiates the processes *Buyer* and *Store*. **Messages**: are typed units of information exchanged over channels and declared by the keyword *mtype*. For example, *mtype*= {*req, res*} declares two types of messages. **Channels**: are the communication medium and declared as local or global by the keyword *chan*. For example, *chan Buyer2Store= [2] of* {*req*} declares a channel that can store up to two messages of type *req*. **Variables**: can be declared as local or global. Typical declarations are *bool flag; byte msg; int counter*. **Send/Receive operations**: are used for sending and receiving messages through channels, and are represented by; *!*, and *?*, respectively. For example, the expression *Buyer2Store ! req(reqNum)* can be used by a *Buyer* process for sending a message to a *Store* process, through the *Buyer2Store* channel. The message is composed of two fields: *req* (the type of message) and *reqNum* (a basic data type such as byte, integer, bool, etc.). To receive the message, the *Store* process can use *Buyer2Store ? req(reqNum)*. **Selection**: *if–fi* is a selection constructor. Every option is guarded. For example, *if :: (debt==0); res=YES :: (debt!=0); res=NO*, will select the appropriate executable statement and render the variable *res* equal to either *YES* or *NO*. **Repetition**: is expressed by *do–od*. The *break* or *goto* statements ae used to terminate the repetition. **Random selection and blocking**: when more than one of the guards evaluates to true in *if–fi* and *do–od* constructs, one of them is selected randomly. If none of them evaluated to true, the process blocks. **Atomic sequences**: two or more instructions enclosed within an *atomic* block are executed as an indivisible unit, that is, in non–interleaved mode.

When a PROMELA model has been created using the above constructs, SPIN can be used for mechanically verifying whether the model satisfies or violates a set of correctness properties.

C. BPMN to PROMELA Mapping

In our work, each BPMN construct is mapped into none, one or several PROMELA constructs. The table in Fig. 3 summarizes the correspondence between BPMN and PROMELA constructs that we use in the BPMN to PROMELA translation process.

BPMN construct	PROMELA construct	Explanation
start event	global variables, global channels, processes	a start event is mapped into several PROMELA constructs
end event	nothing	no mapping needed
activity	channel, message sending and receiving statements	an activity maps into several PROMELA constructs
participant sender	process	a participant is mapped into a single process regardless of the number of activities it participates in
participant receiver	process	a participant is mapped into a single process regardless of the number of activities it participates in
exclusive split gateway	if - fi block	indicates the start of an if - fi block
exclusive merge gateway	nothing	indicates end of an if - fi block
message	mtype message definition	defines a global message
sequence flow	nothing	indicates execution sequence inside a process

Figure 3. Correspondence between BPMN and PROMELA constructs.

D. Linear Temporal Logic

SPIN can mechanically verify the logical soundness of a given PROMELA model presented as input. Conventional safety and liveness properties such as absence of deadlocks and presence of unexpected messages are verified by default by SPIN, so they do not need to be explicitly specified by the designer. However, application–specific correctness properties (such as *payment message is eventually followed by delivery message*) need to be explicitly specified and included in the PROMELA model (one at time) before being presented to SPIN for verification. These correctness properties are specified as Linear Temporal Logic (LTL) formulae. LTL is a formalism proposed for the specifications of correctness properties of concurrent systems [5].

An LTL formula is a logical expression that includes logical variables and unitary and binary operators. The unitary operators are `[]` (always), `<>` (eventually) and `!` (logical negation). The binary operators are `U` (strong until), `&&` (logical and), `||` (logical or), `->` (implication) and `<->` (equivalence). LTL formulae can be conveniently used for expressing correctness properties of choreography diagrams. The procedure involves the edition of the PROMELA model that represents the choreography diagram to include the LTL of interest directly inline within the PROMELA model.

To appreciate the use of LTLs, imagine that a choreography designer wishes to validate some correctness properties of the BPMN choreography diagram of Fig. 2. For example that he would like to be assured that *BuyConf message is eventually followed by either BuyPay or BuyCanc*. To verify this property the designer first needs to express the correctness requirement as an LTL formula for example: `[](c -> <>(p || n);` where c, p and n are propositional symbols. The designer then needs to map these symbols on to boolean variables in the PROMELA code (for example, *confRcvd*, *payRcvd*, *cancRcvd*). These boolean variables are set initially to *false* and become *true*, respectively, when the messages *BuyConf*, *BuyPay* and *BuyCanc* are received. Once this is done, the verifier SPIN can be instructed to check if the model satisfies the LTL property. SPIN produces *number of error: 0* if the LTL property is satisfied or a counter example if it is violated.

Correctness Requirement	LTL formulae
Is buy request (b) eventually sent?	*(<> b)*
Is payment (p) eventually sent?	*(<> p)*
Is buy request (b) eventually followed by either reject (r) or confirmation (c)?	*([ ] b → <> (r || c))*
Is confirmation (c) eventually followed by either payment (p) or cancellation (n)?	*([ ] c → <> (p || n))*
Is payment (p) sent after the request is rejected (r)?	*([ ] r → [ ] !p)*
Is confirmation (c) sent after the request is rejected (r)?	*([ ] r → [ ] !c)*

Figure 4. Examples of correctness requirement and their LTLs.

Constructing such LTLs within a PROMELA model correctly is a challenging task, and especially so as more complex LTL formulae are needed. We propose that this task can be greatly eased by presenting the choreography designer with pre-defined typical LTL templates that can be easily selected and parameterized as needed.

E. Typical LTLs for Choreography Diagrams

A choreography diagram specifies business interactions at the message level and determines the permissible message sequences that business partners are expected to exchange. Although the specific message sequences depend on the application, it is widely acknowledged that there are commonly occurring correctness requirements. To illustrate, the table in Fig. 4 shows some example typical correctness requirements a designer would like to verify against the choreography diagram of Fig. 2. The figure defines the correctness requirement in English and then in LTL.

In Fig. 4, one can observe that some correctness requirements (first and second, third and fourth, fifth and sixth), follow a common LTL pattern. For example, the only difference between the third and fourth LTLs are the names of the prepositional symbols. It follows that these LTL formulae can be mapped onto LTL templates with abstract propositional symbols that can be parameterised to express specific LTL properties.

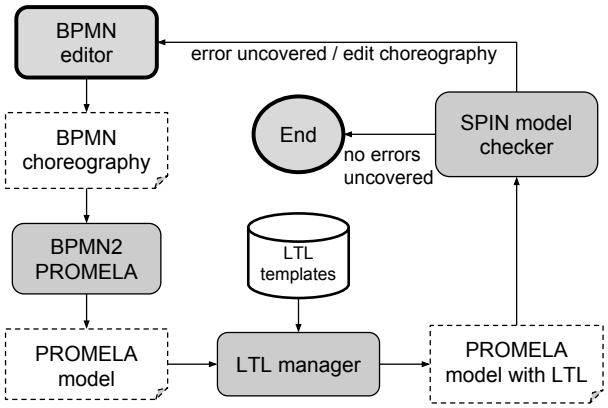

Figure 5. Functional view of the BPMNverifier.

III. Functionality of the BPMNverifier

The BPMN verifier is a software tool implemented in Java, for assisting choreography designers in the verification of choreography diagrams written in BPMN 2.0. A conceptual view of the BPMNverifier is shown in Fig. 5. The main components of the tool are: the BPMN editor, BPMN2PROMELA translator, LTL manager, and the SPIN model checker. Although these components share some data structures created in memory (for example, some java objects) their functionality is independent.

A. BPMN editor

A choreography digram like that of Fig. 2 can be created with the help of a BPMN 2.0 compliant editor. There are several of them available. In our experiments, we have used choreographies produced by the Eclipse BPMN2 modeler that is bundled with the Savara Eclipse tools and freely available from its home page [10]. The BPMN2 Modeler is part of JBoss Savara project [11]. At a lower level, a BPMN choreography diagram is a conventional XML file.

B. BPMN2PROMELA translator

The BPMN2PROMELA translator is capable of automatically translating BPMN 2.0 compliant choreography diagrams to PROMELA models. Fig. 6 gives a general overview of the translation, and shows that each participant in a choreography diagram is translated into a PROMELA process. In this particular example, choreography participants *Buyer* and *Store* are translated into *proc Buyer* and *proc Store*, respectively, communicating by two channels (*B2S* and *S2B*). The *init* process is not shown in the figure.

The BPMN2PROMELA translator includes a configuration file that allows choreography designers to tune some translation parameters to specific needs, such as the size of channel buffers, the disk location of the input (BPMN choreography diagram) and output (PROMELA model) files. With the current version, these parameters need to be edited manually and directly on a text file, at pre–deployment time.

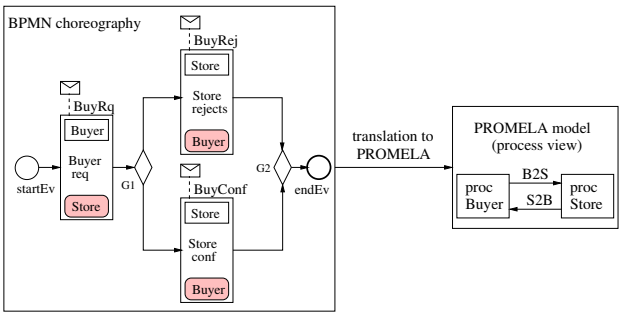

Figure 6. Automatic translation of BPMN choreography to PROMELA.

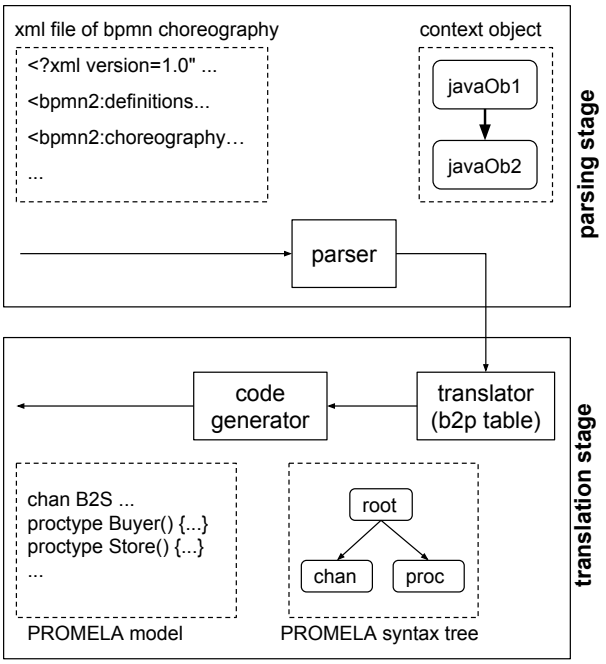

Figure 7. Two stage translation process.

We are planning to include configuration facilities from the main menu of the BPMNverifier in the future. As shown in Fig. 7, the translation from BPMN to PROMELA is based on the conventional two stage translation process: parsing and translating. Intuitively speaking, the parsing stage is concerned with the extraction of the BPMN constructs (events, activities, participants, etc.) from the xml file that represents the choreography; whereas the translation stage deals with the mapping of BPMN constructs to PROMELA constructs. Conceptually, the translation procedure is as follows:

1) *xml file of bpmn choreography* is the xml file produced by the BPMN editor. It is presented as input to the *parser*.
2) The *parser* is a syntactic analyzer that identifies the BPMN constructs (for example, events, activities and participants), and their relationships included in the *xml file of bpmn choreography*, converts them into java

objects ($javaOb_1$, $javaObj_2$, etc.), and stores them in the *context java object*.

3) The *context object* is a memory data structure that stores information about BPMN constructs and their relationships. In the figure for example, $javaOb_1$ and $javaObj_2$ might represent two BPMN activities where the execution of $javaOb_1$ leads to the execution of $javaOb_2$.
4) *b2p table* is a copy of the table in Fig. 3 kept in memory by the *translator*.
5) The *translator* is a semantic analyzer that maps BPMN constructs to PROMELA constructs (channels, processes, if–fi blocks, skip, etc.) following the *b2p table*. It reads java objects from *context object* and outputs them into the *PROMELA syntax tree*.
6) *PROMELA syntax tree* is a memory data structure that contains the PROMELA constructs and information about their relationships.
7) *code generator* is responsible for generating the *PROMELA model*. It is based on a conventional tree traversal algorithm that visits each node of the *PROMELA syntax tree*, identifies the PROMELA constructs, and outputs them into the *PROMELA model*.
8) *PROMELA model* is a plain ascii file that contains a syntactically legal PROMELA model but without any LTL property included. In principle, this PROMELA model can be stored on disk, yet with the current verison of the BPMNverifier, the PROMELA model is kept in memory for the benefit of the *LTL manager*.

C. LTL Manager

The *LTL manager* can be regarded as a graphical interface that can help choreography designers include LTL correctness properties in PROMELA models produced by the BPMN2PROMELA translator. It was implemented in Java and is a core component of the BPMNverifier.

The *LTL manager* offers designers edition capabilities for editing LTL templates (LTL formulae with abstract variables), and stores them in a database (*LTL templates* in Fig. 5). Thus *LTL templates* is a repository of typical LTL formulae collected by LTL experts, and is at the disposition of choreography designers. The database needs to be initialized with some tables before running the BPMNverifier. With the current version of the BPMNverifier, we use Oracle MySql Server 5 [12]. Once the LTL repository has been populated with LTL templates, a choreography designer can retrieve an LTL template of interest, parameterize, and include it in a PROMELA model.

The GUI offered by the *LTL manager* allows designers to select several LTL properties for verification against a given PROMELA model. Note that SPIN can verify only one LTL at a time. Thus the *LTL manager* creates as many instances of the PROMELA model as necessary and invokes SPIN accordingly to verify each PROMELA model separately. For example, imagine that the designer selects three LTL properties (p_1, p_2, p_3) and prompts the *LTL manager* to validate the model against those properties. The *LTL manager* will create a PROMELA validation model with p_1 included, store it on disk, invoke SPIN to validate the PROMELA model, and display the results on the eclipse console. Then it will repeat this procedure for p_2 and p_3.

```
proctype Buyer(){
Buyer2Store ! BuyReq(1);
if
 ::atomic {Store2Buyer ? BuyRej(_);
   BuyRejRcv = TRUE;}
 ::atomic {Store2Buyer ? BuyConf(_);
   BuyConfRcv = TRUE;}
 if
  if
   :: Buyer2Store ! BuyPay(1);
   :: Buyer2Store ! BuyCanc(1);
  fi;
fi;
}
```

```
proctype Store() {
atomic {Buyer2Store ? BuyReq(_);
       BuyReqRcv = TRUE;}
if
::Store2Buyer ! BuyRej(1);
::Store2Buyer ! BuyConf(1);
 if
 ::atomic{Buyer2Store ? BuyPay(_);
   BuyPayRcv = TRUE;}
 ::atomic {Buyer2Store ? BuyCanc(_);
   BuyCancRcv = TRUE;}
 fi;
fi;
}
```

Figure 8. PROMELA model of interaction between buyer and store.

D. SPIN model checker

The SPIN model checker is invoked from the *LTL manager* by the designer. It takes PROMELA models augmented with LTL correctness properties and verifies whether the LTLs are satisfied or violated.

IV. Example

Details about downloading and deploying the BPMNverifier are explained in detail in the tool manual [13]. In this section we will focus only on demonstrating the main features. We will show the verification of some LTL properties from the table in Fig. 4 against the choreography of Fig. 2 whose XML representation is stored is *BuyerStoreChore.bpmn*

1) **Upload the BPMN file:** We use the facilities of the tool to upload the BPMN file (*BuyerStoreChore.bpmn*). The file is stored in the database until it is deleted by the designer, consequently, it can be re–used across sessions. Fig. 8 shows the *buyer* and *store* processes within the PROMELA model generated by the BPMN2PROMELA translator (the full generated PROMELA model has been omitted because of space restrictions).
2) **Edition of LTL templates:** An LTL expert can edit and store LTLs of interest in the *LTL repository* along with their descriptions using the LTL manager (see Fig. 9). The LTL needs to be specified in natural language (*Description* box), and in LTL syntax (*Formula* box). The `@V1@` variable is an LTL propositional symbol that can be parameterized.
3) **Parameterization of LTL formulae:**
Imagine that the designer wishes to validate that the choreography of Fig. 2 satisfies the third and fourth

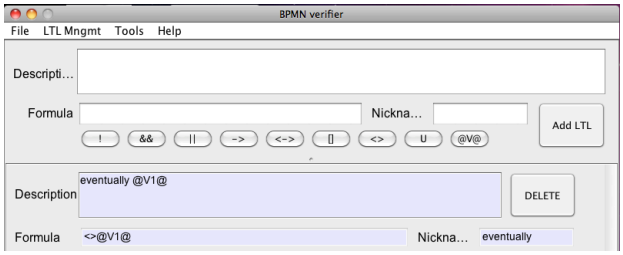

Figure 9. Edition and storage of an LTL template.

properties of the table in Fig. 4. Assuming that an LTL expert has already added the LTL template using the *LTL Manager*, the designer needs to create two instances of the LTL template and parameterize their variables. This is shown in Fig. 10. The tool offers a drop–down list that has all six operations (*BuyReq, BuyRej, BuyConf, BuyPay, BuyCan*) included in the choreography. The designer selects the desired operations as shown in the figure, and the *LTL Manager* automatically creates the correct LTLs.

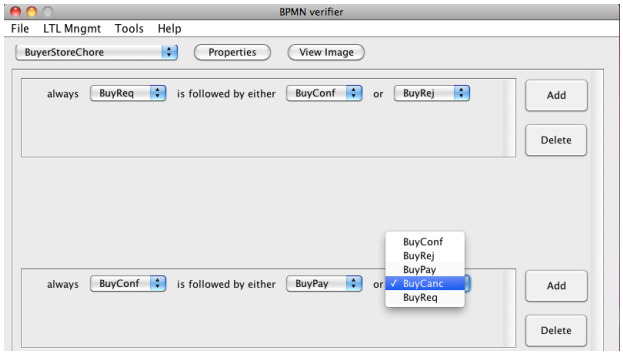

Figure 10. Parameterisation of LTL template.

4) **Validation of PROMELA model:** After the LTL pattern has been parameterized in the previous step, the designer can now simply validate the model by pressing the *Add* button, and then the *Validate button* on the next screen (not shown here). As explained in Section III-C, this action creates two independent PROMELA files —one for each LTL property— that are verified by SPIN.
5) **SPIN validation results:** The results of the validation are displayed within the eclipse console. In this case, both LTLs are satisfied by the validation model; consequently, SPIN displays *errors: 0*.

If on the other hand, the designer adds an LTL property that is violated by the model; for example (`<> BuyPay`) (all execution paths must eventaully result in BuyPay to be executed), SPIN signals that the formulae is violated, and displays *errors: 1*. In addition SPIN creates a trail file in the working folder that can be used by the designer to trace the source of the error within the BPMN model. In this case, an examination of the trail file and of Fig. 2 would reveal that the LTL formula cannot be satisfied because there are execution paths (for example, the one where the *BuyReq* is rejected) that do not include the execution of *BuyPay*. If this is an important requirement, the BPMN designer needs to apply the required corrections to the BPMN model, and then use the BPMNVerifier again to check that the correctness property is now satisfied.

V. Related Work

In [14] the authors suggest the use of mechanical tools for determining whether BPMN choreography specifications are *realizable* as a set of peer processes that communicate with each other observing the global choreography requirements. Tool preferences aside —they use LOTOS NT whereas we use PROMELA— the fundamental difference between theirs and our work is in the approach used for detecting potential flaws. To determine realizability, they rely on the comparison of message sequences produced from an abstract model of the global choreography against message sequences produced from an abstract model of the choreography realised as set of communicating peer processes. They claim that the choreography is sound only if the message sequences match each other. In contrast, in our work, we use only the abstract model of communicating peer processes. We believe that our approach is simpler but requires the choreography designer to select (from the repository of LTL templates) the needed correctness properties to uncover potential logical errors, such as incorrect order of activity execution. We claim that the choreography is sound only if its validation model does not violate the correctness properties expressed in LTL.

The idea of automatically converting a business process model directly to a model checking language is not new. A tool (called *Testbed*) for the verification of business processes, using PROMELA and SPIN is discussed in [15]. The functionality and methodology of *Testbed* is similar to our *BPMNverifier*. However, *Testbed* uses a special purpose business process language (AMBER) whereas the *BPMNverifier* has been developed for a widely used standard language (BPMN). Another difference is that in *Testbed*, LTLs need to be included into the PROMELA model manually, and therefore requiring expertise in Linear Temporal Logic. The authors do suggest that LTLs should be included with the assistance of a graphical interfaces that allow designer to select parameterised patterns of typical LTLs from a drag and drop menu — as we do in our *BPMNverifier* using the *LTL Manager* component.

In [16] the authors suggest that well known business process workflow patterns can be translated into PROMELA. They go on to express interest in building an automatic translator from BPMN to PROMELA as an item for future work. Others such as [17] [18] present interesting approaches on how to convert BPMN to PROMELA, however their work

remains at the theoretical stage, and they have not published any translators to our knowledge yet.

VI. Conclusions and Future Work

We have presented the *BPMNverifier*, a GUI tool that can assist in the verification of choreography specifications written in BPMN 2.0. The tool takes as input an XML file that represents the BPMN choreography and automatically converts it into a PROMELA model. The *BPMNverifier LTL Manager* component, enables the creation and description of common choreography related correctness requirements as LTL templates, which are stored in an *LTL repository*. The choreography designer uses the *LTL manager* to augment the automatically generated PROMELA model with LTL correctness properties that result from the parameterisation of the LTL templates. The PROMELA model can then be presented to the SPIN model checker for verification.

The current version of the *BPMNverifier* provides the main required functionalities identified in this paper. We have tested it with several examples and produced correct results. However, it is still an on–going research tool with room for enhancement at both GUI and functional level.

An issue that needs further exploration is the identification of common correctness requirements that are independent from any particular choreography, and their verification by default. A formal discussion in this direction is presented in [19] where the authors argue that realizable choreographies need to observe the principles of connectedness, well-threadedness, and coherence.

Another item for future work is to extend the functionality of the *BPMN2PROMELA* translator to support a wider subset of BPMN constructs (for instance to handle activities that account for exceptional execution outcomes).

A limitation of the current version of the tool is that the *LTL manager* can manipulate only PROMELA models produced by the *BPMN2PROMELA* translator because the latter presents the PROMELA mode to the former, as a memory data structure (see Fig. 5). This is an unnecessary coupling since the two components are functionally independent, and tools in their own rights. We are planning to decouple them in future versions so that the *LTL manager* can be used for editing PROMELA models irrespectively of their origin.

References

[1] G. Governatori, Z. Milosevic, and S. Sadiq, "Compliance checking between business processes and business contracts," in *10th Int'l Enterprise Distrib. Object Computing Conf. (EDOC'06)*. IEEE CS, 2006, pp. 221–232.

[2] C. Molina-Jimenez and S. Shrivastava, "Establishing conformance between contracts and choreographies," in *IEEE Conf. on Business Informatics (CBI'13)*. IEEE CS, 2013.

[3] G. Salaun, T. Bultan, and N. Roohi, "Realizability of choreographies using process algebra encodings," *IEEE Transactions on Services Computing*, vol. 5, pp. 290–304, 2011.

[4] G. J. Holzmann, *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[5] A. Pnueli, "The temporal logic of programs," in *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, 1977, pp. 46–57.

[6] OMG, "Documents associated with business process model and notation (bpmn) version 2.0," http://www.omg.org/spec/BPMN/2.0, Jan 2011.

[7] RosettaNet, "Rosettanet methodology for creating choreographies," http://www.rosettanet.org, 2012, version Identifier: R11.00.00A.

[8] SPIN, "On-the-fly, ltl model checking with spin," http://spinroot.com, Last accessed 2015.

[9] G. J. Holzmann, "Design and validation of protocols: a tutorial," *Computer Networks and ISDN Systems*, vol. 25, no. 9, pp. 981–1017, Apr. 1993.

[10] T. E. Foundation, "Bpmn2 modeler," http://eclipse.org/bpmn2-modeler, Last accessed 2015.

[11] Jboss, "Savara and testable architecture," http://savara.jboss.org/, Last accessed 2015.

[12] O. Corporation, "Mysql data base," http://www.mysql.com, Last accessed 2014.

[13] W. S. (Jim) and C. Molina-Jimenez, "Deployment of the bpmn verifier (v1.1): Manual," http://homepages.cs.ncl.ac.uk/ellis.solaiman, Jan 2013.

[14] P. Poizat and G. Salaun, "Checking the realizability of bpmn 2.0 choreographies," in *Proc. 27th Annual ACM Symposium on Applied Computing (SAC'12)*, 2012, pp. 1927–1934.

[15] W. Janssen, R. Mateescu, S. Mauw, and J. Springintveld, "Verifying business processes using spin," in *Proc. Spin'98 Workshop*, 1998.

[16] C. Vaz and C. Ferreira, "Formal verification of workflow patterns with spin," Dept. of Electronic and Telecommunications and Computer Engineering ISEL, Polytechnic Institute of Lisbon, Tech. Rep. INESC–ID Tec. Rep. 1212, Apr. 2007.

[17] J. C. P. Aguilar, K. Hasebe, M. Mazzara, and K. Kato, "Model checking bpmn models for reconfigurable workflows," School of Computing Science, Newcastle University, UK, Tech. Rep. CS–TR–1274, 2011.

[18] S. Yamasathien and W. Vatanawood, "An approach to construct formal model of business process model from bpmn workflow patterns," in *Fourth International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*. IEEE, 2014.

[19] M. Carbone, K. Honda, and N. Yoshida, "Structured communication–centered programming for web services," *ACM Transactions on Programming Languages and Systems*, vol. 34, no. 2, Jun. 2012.